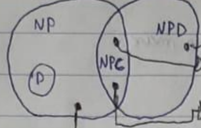
-> Tratabilidade (Em geral, problema de decisão)



- -> reclassificou o problema p/ P, então P = NP = NPC
- -> se for comprovado não polinomial, então P ≠ NP
- -> se encontrar solução polinomial, P = NP = NPC
- -> reclassifica p/ P se encontrar solução polinomial ou mapear p/ um problema em P

PD na verdade

-> Backtracking - top-down é melhor que o bottom-up para matrizes esparsas

Estrutura Básica:

```
int back(estado)
  se estado é solução retorne True
  para cada movimento válido
    efetue o movimento, atualizando o estado
    se back(estado) == True retorne True   // remove o if se quiser TODAS AS SOLUÇÕES
    desfaça o movimento
  retorne False   // não encontrou nenhuma solução
```

Ex: -> 8 rainhas:

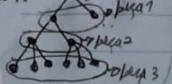
```
int 8Q(int table[8][8], int curr_column)
  if(curr_column==8){print_sol(); return True;}
  for(i=0;i<8;i++){
    verifica_se_ha_rainha_linha(); // pode considerar só colunas anteriores
  if(verifica_se_ha_rainha_diagonal()==0) continue;   // segue essa lógica
    table[i][curr_column] = 1;
    8Q(table, curr_column+1);
    table[i][curr_column]=0; }
    return False;
```

consegue chegar até essa soma

```
int sudoku(table[9][9]){
  int new_row, new_col;
    if(!empty_space(table, &new_row, &new_col))
    {print_table(table); return True;}
    for i in range(1,10){   // 10 not incl.
      if(is_valid(table, new_row, new_col, i)){
        table[nr][nc]=i; sudoku(table);
        table[nr][nc]=0;}
    }
    return False;
}
```

-> Programação Dinâmica

• Peças:



usando até peça 1,2,3,...

	1	2	3	...	18	19	20
1							
2							
3							

tilibra

/ /

- Mochila sem restrições → nenhum dígito soma 0 de peso

Obj	Peso	Valor
1	5	50
2	3	20
3	2	19

m = | 0 | 0 | 19 | 20 | 38 | 59 | 50 | 69 | 70 | 88 | 109 |
(índices 0 1 2 3 4 5 6 7 8 9 10 → peso máx.)

```
weight_max = 10
for(i=0; i<=weight_max; i++){ int max = 0;
    for(j=0; j< size; j++){ if(peso > i) continue;
    if(valor[j] + m[i-peso[j]] > max){ max = valor[j] + m[i-peso[j]]; }
    } m[i] = max; return memoize[weight_max];
```

- Mochila 01 → pega o de cima ou o valor do item da linha somado do valor da memoize na linha atual na coluna atual subtraída do peso do item atual.

V	W		0	1	2	3	4	5	6
		0	0	0	0	0	0	0	0
(10)	(3)	1	0	0	0	10	10	10	10
(8)	(2)	2	0	0	8	10	16	18	24
(15)	(4)	3	0	0	8	10	16	18	24

```
int mochila01(int item, int peso){
    if(item<0 || peso<0) return 0;
    if (m[item][peso] != 0) return m[item][peso];
    if(W[item] > peso) return mochila01(item-1, peso);
    int max = mochila(item-1, peso); int novo = mochila(item-1, peso - W[item]);
    if(v[item] + novo > max) max = v[item] + novo; m[item][peso] = max;
    return m[item][peso]; }
```

- Multiplicação de Matrizes – $M(p,q) \times N(q,r) = O(p,r)$ → $A(p,q) \times B(q,r) \times \cdots \times E(m,n)$

	0	A	B	C	D
0	0	0xA	0xB	0xC	0xD
A		0	AxB	min	min
B			0	BxC	min
C				0	CxD
D					0

0xD → menor de A a C
min → menor de A a D
min → menor de B a D

```
int num_coins() {
    for (i=1; i<= size; i++){
    for (j=1; j<= num; j++){
    int up = m[i-1][j];
    int np = j - v[i-1]; left = 0;
    if(np>=0) { if(np==0 || m[i][np] != 0){
    left = m[i][np]+1; } }
    if (left==0 && up==0) m[i][j] = 0;
    elif (left!=0 && up==0) m[i][j] = left;
    elif (left==0 && up!=0) m[i][j] = up;
    elif (left!=0 && up!=0) min(left, up);
    } return m[size][num]; }
```

- Moeda

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	0	2	0	3
2	0	1	1	2	2	3	3
3	0	1	1	2	1	2	2
4	0	1	1	2	1	2	1

```
int moeda_solve (moedas, num) {
    if(num==0) return 1;
    if(m[mu][num] != -1)
        return m[mu][num];
    curr_sol = moeda(mu-1, num)
    if(num - c[mu-1] >= 0)
    { curr_sol += moeda(mu, num - c[mu-1]); }

    m[mu][num] = curr_sol; return m[mu][num];
```

tilibra